

CO3302 Computer Engineering Project

PC Market Expert

Ranasinghe R.A.C.M.

November 17, 2022

Abstract PC Market Expert is an application programming interface that is built to provide data related to the consumer PC market in Sri Lanka. This API mainly utilizes web scraping which is an automatic method to obtain large amounts of data from websites. In the back end of the system, a web scraper runs periodically and collects data from local consumer PC shops. The extracted data is stored in a database. API fetches this data from the database and serves it to the user. The main goal of this system is to provide a platform for businesses and developers to do market research and analysis (especially competitor monitoring) without using a lot of capital or resources.

1 Introduction

1.1 Background

As an entrepreneur as a business owner or as a person who can impact the organization on a large-scale, market research and analysis is essential to the persistence of a business. Market research and analysis are important for market trend analysis, price monitoring, research and development, and competitor monitoring. Data is mainly needed to do effective market research. Since the internet is growing continuously, the amount of data it generates is also growing. Incorporating this large amount of data for market research and analysis can ensure that the organization continuously grows without any disruptions[8].

In the context of the consumer PC market in Sri Lanka, market research and analysis are crucial for small and large businesses in terms of gaining customers while staying competitive in the market. Market research and analysis can be used when making decisions about product pricing and selecting the products to sell.

1.2 Problem Definition

However, acquiring data manually for market research is an arduous and mundane task. It can be time-consuming, expensive, and human resource-consuming. This makes the whole process considerably inefficient and prone to errors. Especially for small-scale businesses, which do not possess the capital to spend on that level of market research. Even if the data acquisition needs to be automated those businesses lack the capital to create applications that are suited for the task. To create such an application developers will need data regarding the consumer PC market in Sri Lanka.

2 Related Work

In the context of the consumer PC market in Sri Lanka, there are no applications that provide data about the market. Data obtaining must be done manually using human resources. When compared with manual data acquisition developers can use this API to automate this tedious task and create applications that can be used for market research and analysis (predictive analysis, neural networks, statistical analysis, etc.). These applications do not need to be focused only on businesses and business owners, consumers can also use these applications to find products, etc.

3 Objectives

The main objectives of the system are as follows.

- Provide necessary data for e-commerce product research and market analysis in the consumer PC market in Sri Lanka.
- Create a fully autonomous data scraping system for the PC market in Sri Lanka.
- Provide an API for developers to utilize the data regarding consumer PC market.

4 Methodology

To accommodate for constant changes and to improve the quality of the overall system agile development method was utilized.

Following environments were used in the process of system development.

- Local Development Environment
 - Intel® Core™ i5-9300H CPU @ 2.40GHz
 - 16 GB Memory
 - Windows 11 + Ubuntu 20.04 (WSL - Windows Subsystem for Linux)
- Testing Environment
 - GitHub Actions (Ubuntu 22.04)

- Deployment Environment
 - Heroku (virtualized Unix container)

4.1 Requirement Analysis

To define the user needs of the application and to avoid future problems requirement analysis was done at the beginning of the development process. After the requirement analysis stage following requirements were identified.

- The application must be able to serve user requests in real-time or near real-time.
- The application must maintain consistency with data that is gained from different sources. Data must be prepossessed and cleaned before serving into the users.
- Since the market is always changing, the application must be able to be continuously updated.
- The application must provide data according to a standard thus that developers can easily work with data.
- Documentation that defines the system and its parts must be provided to users thus users can easily integrate the API into their applications and troubleshoot in case of an error.
- Data gathering (scraping and pre-processing) in the application must be done efficiently to provide updated data as soon as possible.
- The application must provide data regarding the products available in different shops, product price history, and statistics of the products available in each shop.

4.2 System Design

Mainly two designs were chosen in the process of system design.

1. When a user requests data, web scraping can be done in real-time and provide data after preprocessing.
2. Web scraping is done periodically, and the data is stored in a database. When a user requests data, data is fetched from the database.

According to requirements, data should be provided in real-time or near-real-time. In the first approach, this cannot be guaranteed since sending a request to a web page and scraping data can take time which makes real-time responding hard. A considerable amount of resources will also be needed in the first approach. Moreover, if some users request the same data, the same web page will be scraped again and again making the whole process inefficient. The advantage of this design is that updated data can be provided as soon as possible since scraping is done in real-time.

In the second approach since the data is already in the database, data can be provided in real-time or near-real-time. Compared to the first approach less amount of resources are needed and the process is efficient since one web page will be scraped only once. One disadvantage of this design is that updated data cannot be provided as soon as possible since web scraping is done periodically. But in the context of the consumer PC market, this is not a problem since new products are not added regularly.

Thus, considering the above factors second design was chosen and the following architecture in Figure 1 was created.

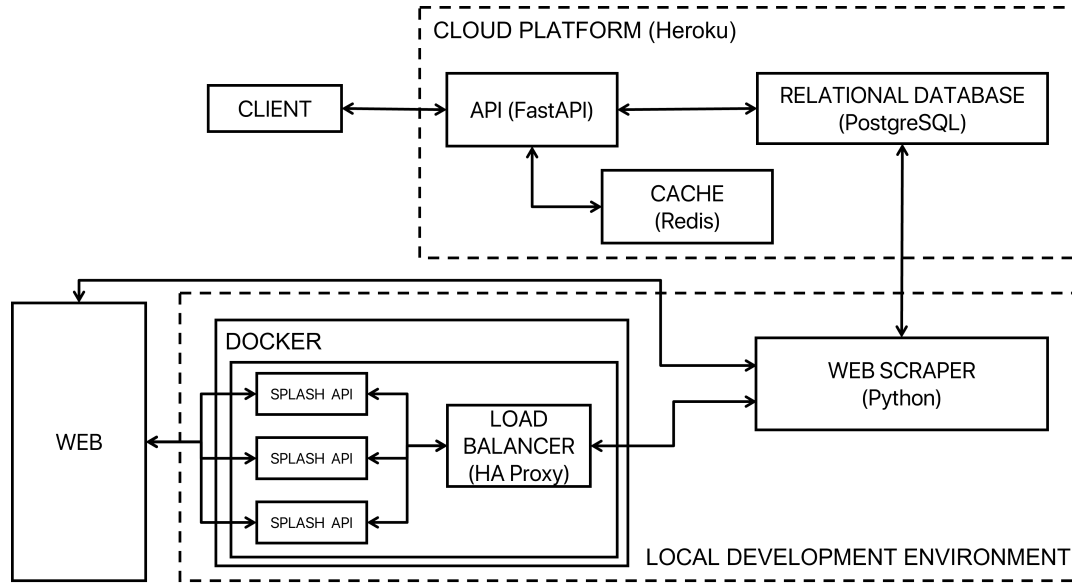


Figure 1: Overall architecture of the system

4.2.1 Architecture

The architecture mainly consists of three parts.

- **Application Programming Interface**
API will serve as the main interface to access data. Once a user request data, after validating the request API will fetch the data from the database, after again validating the fetched data, API will serve the data to the user.
- **Database**
The database will store all the data that is generated after the web scraping process.
- **Web Scraper**
The web scraper will be responsible for scraping data, preprocessing, and cleaning. After the web scraping is done data will be stored in the database. Web scraping will be done periodically depending on the task.

Other than that, cache and a javascript rendering system are also utilized in the system. The cache is used to increase performance by minimizing calls to the relational database. The JavaScript rendering system is used to convert dynamic web pages to static web pages by executing the javascript in web pages on the client side[16].

Although initially it was purposed to run the whole system in the cloud fully automated. The javascript rendering system (Splash API cluster) required a considerable amount of infrastructure to execute. Thus, along with that web scraper was not deployed in the cloud.

4.2.2 Application Programming Interface

To access the API users will need to obtain an API key. A public route in API is available to users to obtain the API key[6]. Users will need to send their email addresses in a POST request, to obtain a valid API key. Once the API key is given, it will be hashed and stored in the database along with the user's email address. When a user sends requests to a secured route user must include the API key in the headers. After validating the API key users can retrieve data from the API. Although the API is public this functionality was added to add a basic layer of security on the client side.

Schemas for responses and requests are predefined in the API. Thus, request parameters are validated before fetching data from the database, and response data is validated before serving to the user.

To increase the performance of the API caching layer is also added. Since web scraping is done periodically, data in the database is not updated continuously thus responses for some routes are cached to expire within a day (since web scraping is done once per day). Also, when the server is starting API keys are cached to improve the performance.

4.2.3 Web Scraper

Architecture in the Figure 2 was used in the web scraper. Web Scraping is mainly done in two parts;

1. Scraping product data on computer shop websites.
2. Scraping reference data on other third-party websites.

Reference data scraping is done once a week, and product data scraping is done once a day. At first, the web scraper generates an URL list containing web pages that need to be scrapped. After that web scraper requests the web page from websites. Once the web page (HTML) arrives, it is parsed, and an element tree is built. This is done to navigate the page to extract data. In reference-data scraping after extracting and cleaning the extracted data is stored locally. In product data scraping after scraping, the data is provided into a product matching algorithm to identify the product according to the reference data. Once it is identified other details (main specifications of the product) are fetched from the reference data.

The product matching algorithm is mainly based on Levenshtein distance. The Levenshtein distance is a string metric for measuring the difference between two sequences.

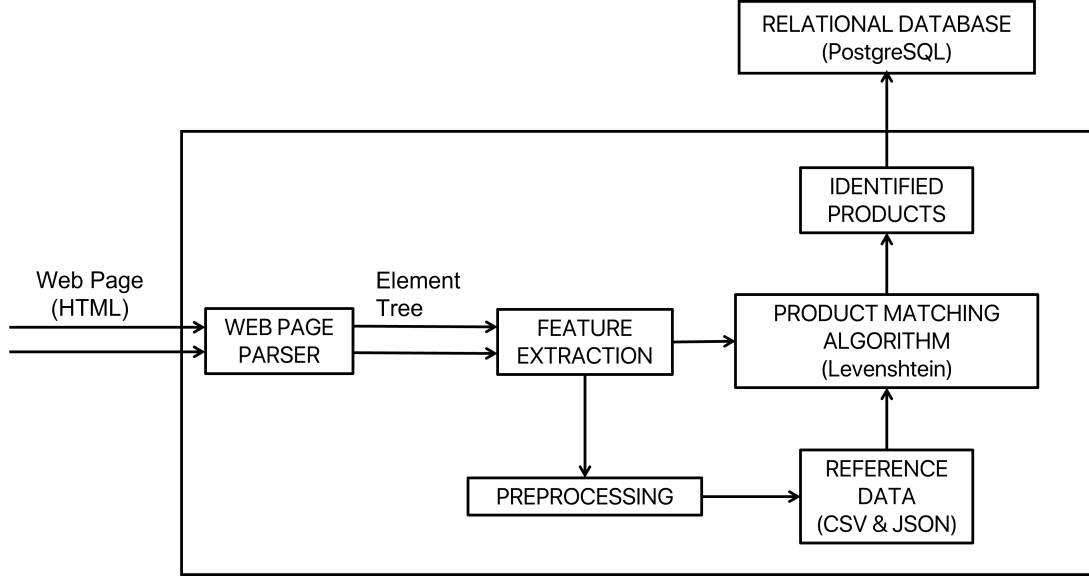


Figure 2: Overall architecture of the web scraping system

Informally, the Levenshtein distance between two words is the minimum number of single-character edits (i.e., insertions, deletions, or substitutions) required to change one word into the other[reference]. For different products the overall algorithms are different. This is done to increase the accuracy of product matching. Finally, the extracted product data is stored in the database.

4.2.4 Database

The database is mainly responsible for storing data that is scraped from the web scraper and API keys with the relevant email addresses. To accommodate complex querying and to maintain a structured database relational database is used.

4.3 Implementation

Implementation was mainly done using Python programming language. Since the system is based on data analysis Python was used. Python as a language provides good support (libraries, frameworks) for tasks related to data analysis. Other following tools and libraries were used to build the system are listed below.

4.3.1 API

- FastAPI

FastAPI is a python framework for building APIs. FastAPI was chosen because of its performance, development speed, robustness, standard, and intuitiveness.[13]

- SQLAlchemy

SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL. It provides a full suite of well-known enterprise-level persistence patterns, designed for efficient and high-performing database access, adapted into a simple and Pythonic domain language.[5]

- Postman

API development and management platform[12].

- Redis

Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache, and message broker[3].

4.3.2 Web Scraper

- BeautifulSoup

Beautiful Soup is a Python library for pulling data out of HTML and XML files. Works with many parsers and provide ways to search for data.[15]

- Aiohttp

Asynchronous HTTP Client/Server for asyncio and Python.[9]

- Requests

HTTP library for Python [14].

- FuzzyWuzzy

Python library that can be used for string matching using Levenshtein Distance[7].

- cProfile, pstats, Snakeviz

4.3.3 Cloud Platform

- Heroku

Heroku is a platform as a service (PaaS) that enables developers to build, run, and operate applications entirely in the cloud[1].

4.3.4 Database

- PostgreSQL

Free and open-source relational database management system[2].

- PgAdmin

Free and open-source administrative development platform for PostgreSQL[11].

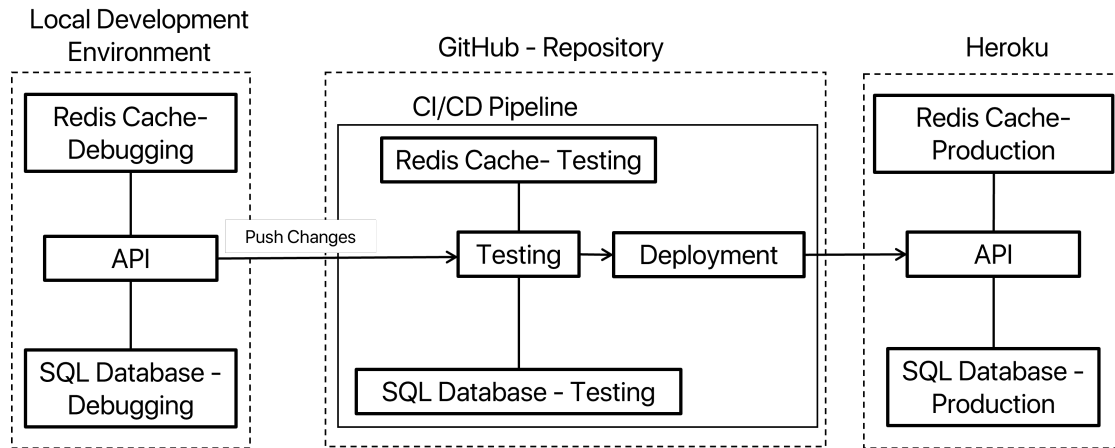


Figure 3: Development Process

4.4 Testing

Unit testing was done in the early stage of the system development process. Unit testing was mainly done using Postman. After integrating the web scraper, database, and API integration testing was done. In the early stage of the system development process, integration testing was done manually. Further to the development process integration testing was automated using the CI/CD pipeline as in Figure 3. Testing was mainly done using the Pytest library.

4.5 Deployment

In the early stage of the development Deployment to the cloud platform was done automatically using the CI/CD pipeline. Git and GitHub were used for version control. GitHub Actions was used to create the CI/CD pipeline as in Figure 3.

5 Results

5.1 Functionalities

- Provide data about products

API provides data about product specifications, availability, and price. Also, users can query data based on category and specification.

- Provide data shop-wise.

Provide product data. Also, provide statistical data about the products in the shop category-wise. Statistical data is mainly based on the number of products and availability of products as in Figure 5.

- Provide historical data on products.

Provides data on past days. Mainly availability and price are provided.

Currently, API supports three popular local consumer PC shops (Nanotek, Tech-Zone, Gamerstreet) and six product categories (processors, graphic processing units, motherboards, memory, storage, and power supplies). All the data is provided in JSON format in real-time.

Figure 4 shows one JSON object that returns to a query to find processors with a core count of 6, a thermal design power (TDP) of 65 Watts, and contains an integrated graphics processing unit. Depending on the result, an array of these matching objects are returned for a query.

A Sample web application was also developed for better visualization of data provided by the API using HTML, CSS, and JavaScript as in Figure 6.

```
{
  "id": 3947,
  "name": "amd ryzen 5 5600g",
  "prices": {
    "gamestreet": 80000.0,
    "nanotek": 72000,
    "tech-zone": 83500.0
  },
  "category": "cpu",
  "brand": "amd",
  "links": {
    "gamestreet": "https://www.gamestreet.lk/product_view.php?pid=MTMyOQ==",
    "nanotek": "https://www.nanotek.lk/product/1139",
    "tech-zone": "https://techzone.lk/product/amd-ryzen-5-5600g-with-radeon-graphics/"
  },
  "shops": {
    "gamestreet": "amd ryzen 5 5600g (6 cores, 12 threads) up to 4.4 ghz desktop processor with wraith stealth cooler",
    "nanotek": "amd ryzen 5 5600g",
    "tech-zone": "amd ryzen 5 5600g with radeon graphics"
  },
  "availability": {
    "gamestreet": true,
    "nanotek": true,
    "tech-zone": true
  },
  "specs": {
    "core-count": 6,
    "performance-core-clock": 3.9,
    "performance-boost-clock": 4.4,
    "integrated-graphics": "Radeon Vega 7",
    "smt": "Yes",
    "tdp": 65
  },
  "index": 6
}
```

Figure 4: Example response for product data query.

5.2 Performance

Performance is a key requirement of the system. Since FastAPI is used as the framework API has relatively high performance compared to other APIs built with other Python

```

{
  "shop": "gamestreet",
  "numberOfProducts": {
    "motherboard": 95,
    "power-supply": 29,
    "storage": 23,
    "gpu": 16,
    "cpu": 49,
    "memory": 24
  },
  "availabilityOfProducts": {
    "motherboard": 28,
    "power-supply": 6,
    "storage": 3,
    "gpu": 1,
    "cpu": 17,
    "memory": 6
  }
}

```

Figure 5: Example response for shop metadata query.

frameworks such as Flask or Django. Since FastAPI is based on Starlette and Uvicorn, API performs relatively better.[13]

The performance of the web scraper mainly depends on networking since networking is I/O bound. Web page processing (scraping) performs much better due to optimizations done in the development process. Some of them are listed below.

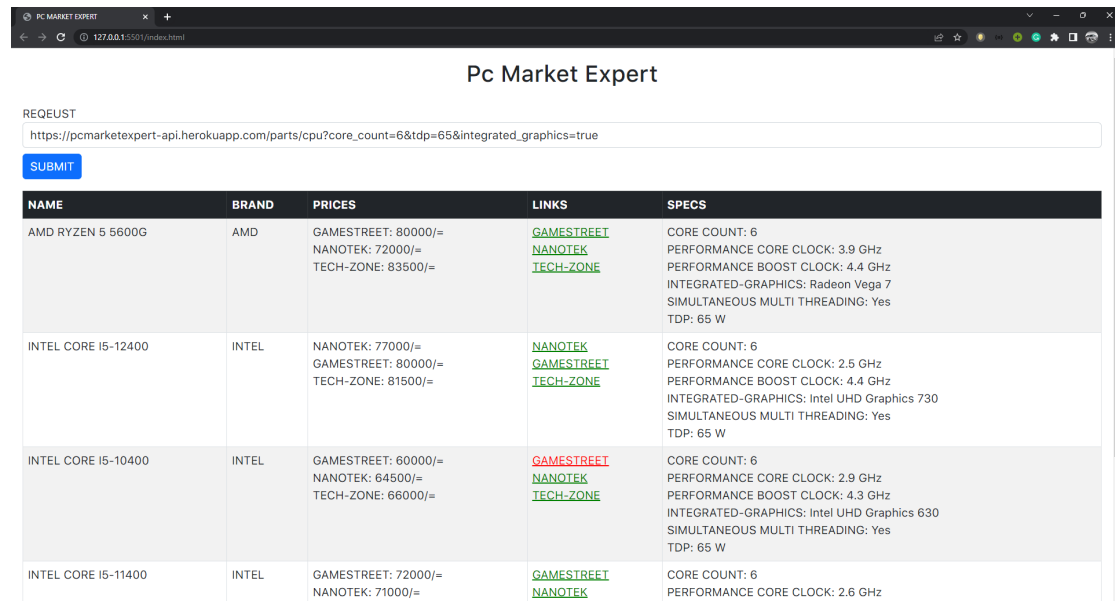
- Parsing relevant sections of the web page rather than parsing the whole web page. Although this does not reduce much parsing time. This saves considerable amount of memory and speeds up the searching in the document.
- Using LXML parser instead of HTML parser.
- Using cchardet library to speed up encoding detection.

5.3 Scalability

The scalability of the system mainly depends on the scalability of the web scraper. In the stage of implementation, the code was organized in a way such that adding support to new websites and product categories can be done without much effort. The scalability of the API and database mainly depends on the infrastructure provided by the cloud platform.

5.4 Documentation

FastAPI framework provides the support to generate documentation. This documentation is interactive and provided by Swagger UI. Anyone willing to try the system can interactively send requests and analyze results without any preparation. Documentation provides a comprehensive description of the routes and schemas. Description for routes was added in the code so that FastAPI autogenerates the documentation.



PC MARKET EXPERT

REQUEST

https://pcmarketexpert-api.herokuapp.com/parts/cpu?core_count=6&tdp=65&integrated_graphics=true

SUBMIT

| NAME | BRAND | PRICES | LINKS | SPECS |
|---------------------|-------|---------------------|----------------------------|---|
| AMD RYZEN 5 5600G | AMD | GAMESTREET: 80000/= | GAMESTREET | CORE COUNT: 6 |
| | | NANOTEK: 72000/= | NANOTEK | PERFORMANCE CORE CLOCK: 3.9 GHz |
| | | TECH-ZONE: 83500/= | TECH-ZONE | PERFORMANCE BOOST CLOCK: 4.4 GHz |
| | | | | INTEGRATED-GRAPHICS: Radeon Vega 7 |
| | | | | SIMULTANEOUS MULTI THREADING: Yes |
| | | | | TDP: 65 W |
| INTEL CORE I5-12400 | INTEL | NANOTEK: 77000/= | NANOTEK | CORE COUNT: 6 |
| | | GAMESTREET: 80000/= | GAMESTREET | PERFORMANCE CORE CLOCK: 2.5 GHz |
| | | TECH-ZONE: 81500/= | TECH-ZONE | PERFORMANCE BOOST CLOCK: 4.4 GHz |
| | | | | INTEGRATED-GRAPHICS: Intel UHD Graphics 730 |
| | | | | SIMULTANEOUS MULTI THREADING: Yes |
| | | | | TDP: 65 W |
| INTEL CORE I5-10400 | INTEL | GAMESTREET: 60000/= | GAMESTREET | CORE COUNT: 6 |
| | | NANOTEK: 64500/= | NANOTEK | PERFORMANCE CORE CLOCK: 2.9 GHz |
| | | TECH-ZONE: 66000/= | TECH-ZONE | PERFORMANCE BOOST CLOCK: 4.3 GHz |
| | | | | INTEGRATED-GRAPHICS: Intel UHD Graphics 630 |
| | | | | SIMULTANEOUS MULTI THREADING: Yes |
| | | | | TDP: 65 W |
| INTEL CORE I5-11400 | INTEL | GAMESTREET: 72000/= | GAMESTREET | CORE COUNT: 6 |
| | | NANOTEK: 71000/= | NANOTEK | PERFORMANCE CORE CLOCK: 2.6 GHz |

Figure 6: Web Application

6 Challenges

- Dynamic web pages

To extract data from web pages, the relevant page needs to be static. But some web pages contain Javascript code that needed to be executed in the client-side browser. Simply requesting a web page does not work in this scenario. To accommodate that Splash API was used, which is a Javascript rendering system. To accommodate a large number of requests cluster of this API was used along with a load balancer (HaProxy). This was configured using Aquarium which is a splash and docker-compose setup.

- Managing requests

Initially asynchronous requests were used to scrape data from the websites[4]. In product data scraping requests are divided between websites, thus sending asynchronous requests is not a problem. But when reference data scraping a large number of requests are sent to a single website[10]. This can result in IP address banning since web servers may this behavior as a cyber attack attempt. On the other hand sending, synchronous requests can significantly decrease the performance of the web scraper. Thus a hybrid system was used, utilizing both asynchronous and synchronous requests.

- Infrastructure constraints

Javascript rendering system was used as a cluster of Splash API along with a load balancer in the docker. A cluster was used to accommodate a large number of

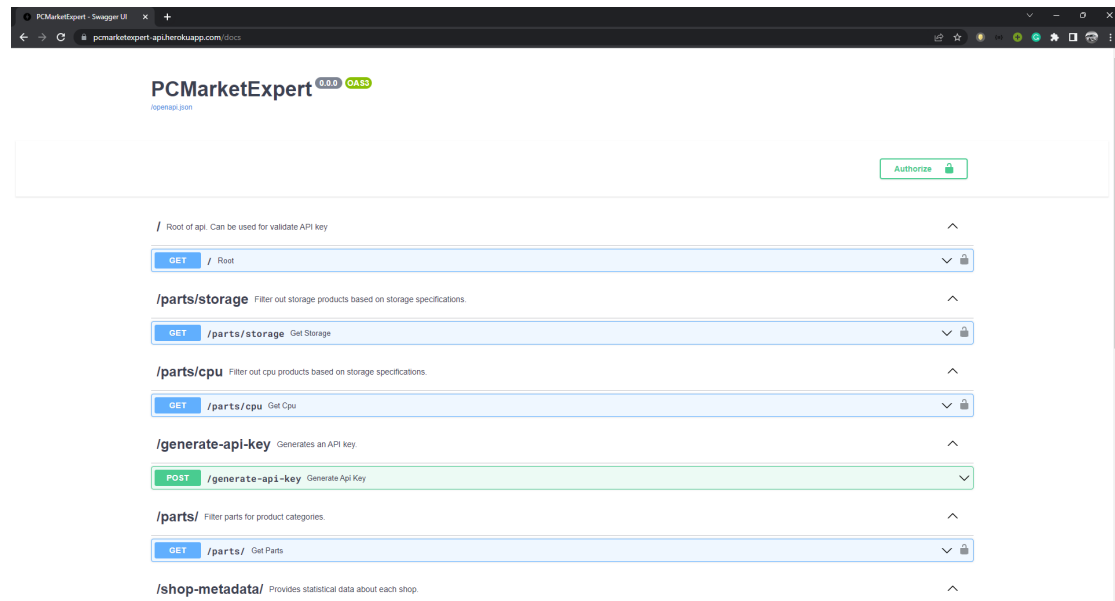


Figure 7: Screenshot of the interactive documentation generated by FastAPI.

requests. If rendering a large number of web pages done by a single instance of the API it will take a significant amount of time. This system required a considerable amount of infrastructure to execute. Thus, this system along with the web scraper was not deployed in the cloud platform as proposed due to infrastructure constraints.

7 Further Improvements

- Deploy the whole system in the cloud with more infrastructure.
- Increase the support for more websites and add more product categories.
- Increase the security of the API.
- Provide more statistical and analytical data related to the consumer PC market in Sri Lanka.
- Improve the product matching algorithm to increase the accuracy of product matching.
- Dockerize the whole system to increase portability and scalability.
- Use data mining to provide valuable insights related to the consumer PC market.

References

- [1] Heroku documentation.
- [2] PostgreSQL documentation.
- [3] Redis documentation.
- [4] Gunardi Ali. Best way to speed up a bulk of http requests in python, Jan 2022.
- [5] Michael Bayer. Sqlalchemy. In Amy Brown and Greg Wilson, editors, *The Architecture of Open Source Applications Volume II: Structure, Scale, and a Few More Fearless Hacks*. aosabook.org, 2012.
- [6] Josh Campos. Securing fastapi with api keys, Apr 2022.
- [7] Adam Cohen. Fuzzywuzzy.
- [8] Colm Kenny. How to use web scraping for market research - zyte, Mar 2022.
- [9] Nikolay Kim and Andrew Svetlov. Aiohttp documentation.
- [10] Yelyzaveta Nechytailo. Asynchronous web scraping with python and aiohttp, Aug 2022.
- [11] Dave Page.
- [12] Postman. Postman documentation, Sep 2022.
- [13] Sebastián Ramírez. Fastapi, 2019.
- [14] Kenneth Reitz, Cory Benfield, Ian Stapleton Cordasco, and Nate Prewitt. Requests documentation.
- [15] Leonard Richardson. Beautiful soup [https://www. crummy. com/software](https://www.crummy.com/software), 2014.
- [16] TeamHG-Memex. Teamhg-memex/aquarium: Splash + haproxy + docker compose.